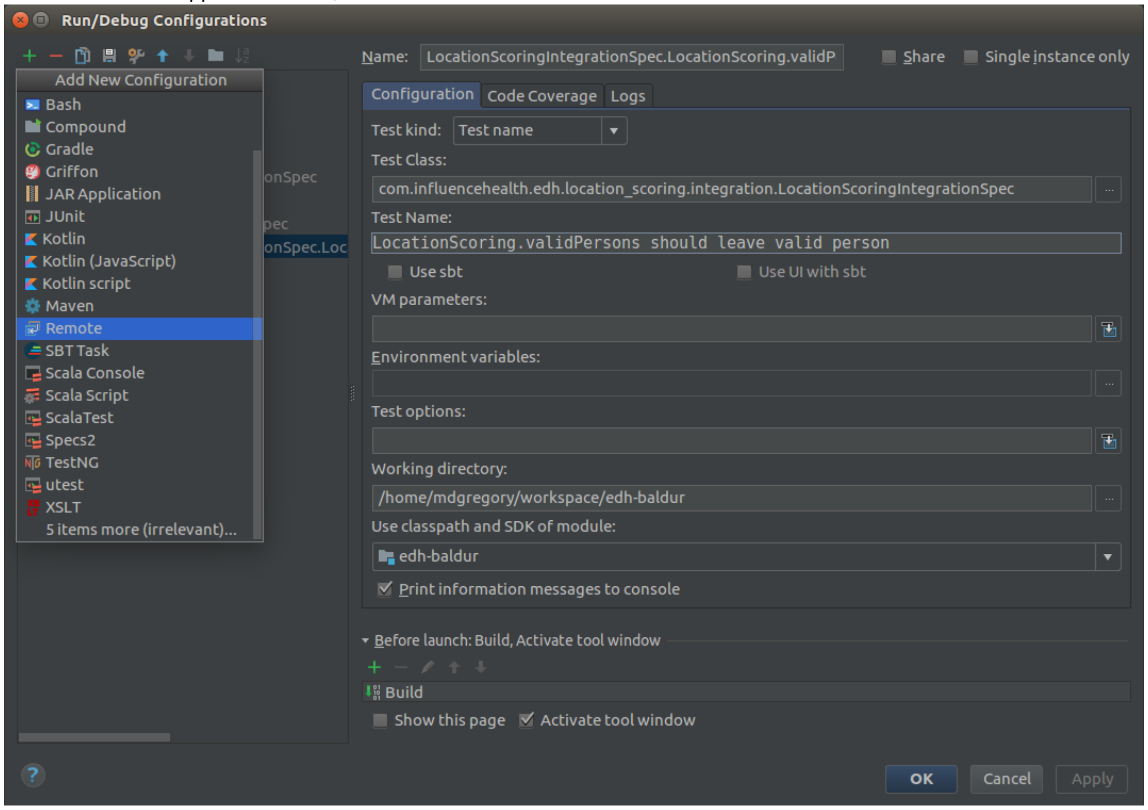
Debugging in IntelliJ

Configuring IntelliJ

1. Open IntelliJ and go to **RunEdit Configurations** menu
2. Click the "+" in the upper left corner, and choose "Remote"



Name: LocationScoringIntegrationSpec.LocationScoring.validP

Test kind: Test name

Test Class:
com.influencehealth.edh.location_scoring.integration.LocationScoringIntegrationSpec

Test Name:
LocationScoring.validPersons should leave valid person

VM parameters:

Environment variables:

Test options:

Working directory:
/home/mdgregory/workspace/edh-baldur

Use classpath and SDK of module:
edh-baldur

✓ Print information messages to console

Before launch: Build, Activate tool window

Build

✓ Activate tool window

3. Edit the configuration name to something meaningful to you, for example "Remote Debugger"
4. Click Apply/OK

Configuring baldur to run in debug mode

1. In the new scripting code, all jobs will be able to make use of a new parameter (this should always be the last parameter passed to the script). the values will be: -***debug*** or -***debugSuspend***
   a. ***--debug*** | ***-d*** = enable java debugging but do not wait for debugger to attach, continue executing as normal
   b. ***--suspend* | *-s*** = enable java debugging but pause until the debugger attaches to the process (can be useful if you have to debug the starting up of a process)
2. Run your process (in this example i'm running a local Experian cleanse process):

```
./bin/edh experian cleanse -e local -b "fixtures/experian" -p small -s
```

Attaching the debugger to the running application

1. After running the script you will see something similar to the following output...

```
Using baldur version 0.5.0
Executing using split assembly against
'target/scala-2.11/baldur_2.11-0.5.0.jar' along with
'target/scala-2.11/baldur-assembly-0.5.0-deps.jar'
Listening for transport dt_socket at address: 5005
```

2. Open up intellij and load the project you are running
3. Set any desired breakpoints (if you haven't already)
4. Go to **RunAttach to local process** and your process should be listed
5. Click the process to attach the IDE to the process and debug as normal

Debugging against Prod and Stage

If you are able to tunnel port 5005 from the environment in question to your local machine you can attach your local debugger to a job running in prod or stage the same way you debug local code with the following steps. ***Note***: *the code is **still** running the prod or stage environment, your debugger just matches the executing job to your un-compiled version locally. If your code that you have checked out locally does not match the same revision that is deployed where the job is running, then you will get some extremely strange behavior (breakpoints not being hit, etc). The executing job has zero knowledge of your local code.*

1. check out the same branch/revision of code that is deployed in the environment running the job. I usually check this out into a completely new workspace to avoid any confusion. Open this project in intellij.
2. in a brand new terminal window, you need to ssh tunnel the debugging port from prod or stage to your local machine. you can use this command ***(note: substitute your username in where mine is "mikegregory" in the example below.)***

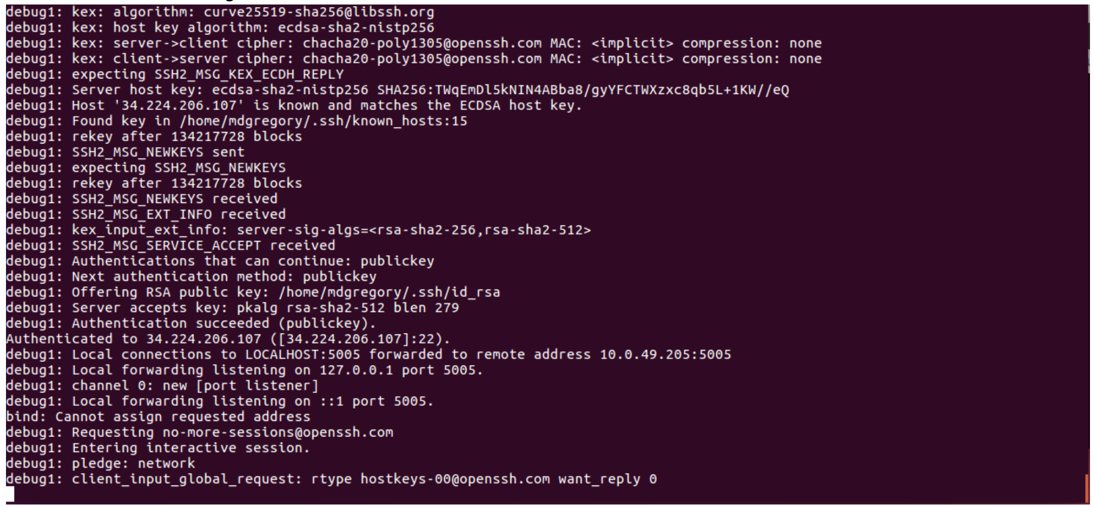
Prod Tunnel Example

```
# this is for PRODUCTION
ssh -v -L 5005:10.132.49.165:5005 -N -l mikegregory 34.225.232.21
```

Stage tunnel example

```
# this is for STAGE
ssh -v -L 5005:10.0.49.205:5005 -N -l mikegregory 34.224.206.107
```

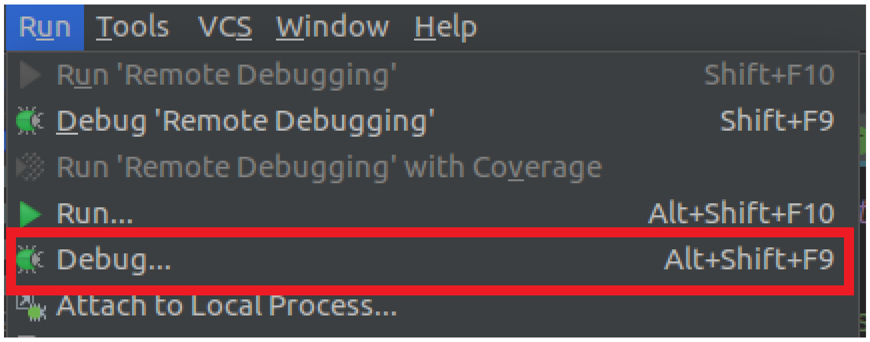
3. after running the above command you should see a bunch of verbose output to the terminal and it should juust appear to "hang" there...itll look something like this:

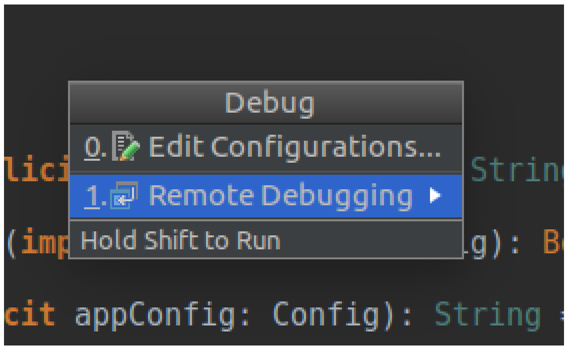


```
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:TWqEmDl5kNIN4ABba8/gyYFCTWXzxc8qb5L+1KW//eQ
debug1: Host '34.224.206.107' is known and matches the ECDSA host key.
debug1: Found key in /home/mdgregory/.ssh/known_hosts:15
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<rsa-sha2-256,rsa-sha2-512>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/mdgregory/.ssh/id_rsa
debug1: Server accepts key: pkalg rsa-sha2-512 blen 279
debug1: Authentication succeeded (publickey).
Authenticated to 34.224.206.107 ([34.224.206.107]:22).
debug1: Local connections to LOCALHOST:5005 forwarded to remote address 10.0.49.205:5005
debug1: Local forwarding listening on 127.0.0.1 port 5005.
debug1: channel 0: new [port listener]
debug1: Local forwarding listening on ::1 port 5005.
bind: Cannot assign requested address
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
```

4. starting the job in prod or stage with the "-s" parameter just as you do for local debugging. it should wait for a debugger to attach before proceeding.

5. in your local intellij, set your break points and attach your debugger by going to "Run" menu and choosing "Debug" and selecting the remote debugger you configured earlier in this guide. Mine is called "Remote Debugging". See below...
6. At this point, the debugger should attach, the job should continue and hit your break points.


Run
Tools
VCS
Window
Help
Run 'Remote Debugging' Shift+F10
Debug 'Remote Debugging' Shift+F9
Run 'Remote Debugging' with Coverage
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Attach to Local Process...



When youare done debugging you <u>can and should</u> end the tunnel by hitting CTRL+C in the terminal where you started the tunnel (in windows, command+c on a mac...i think)